

SSD Parvex SAS

8, avenue du Lac - B.P. 249

F-21007 Dijon Cedex

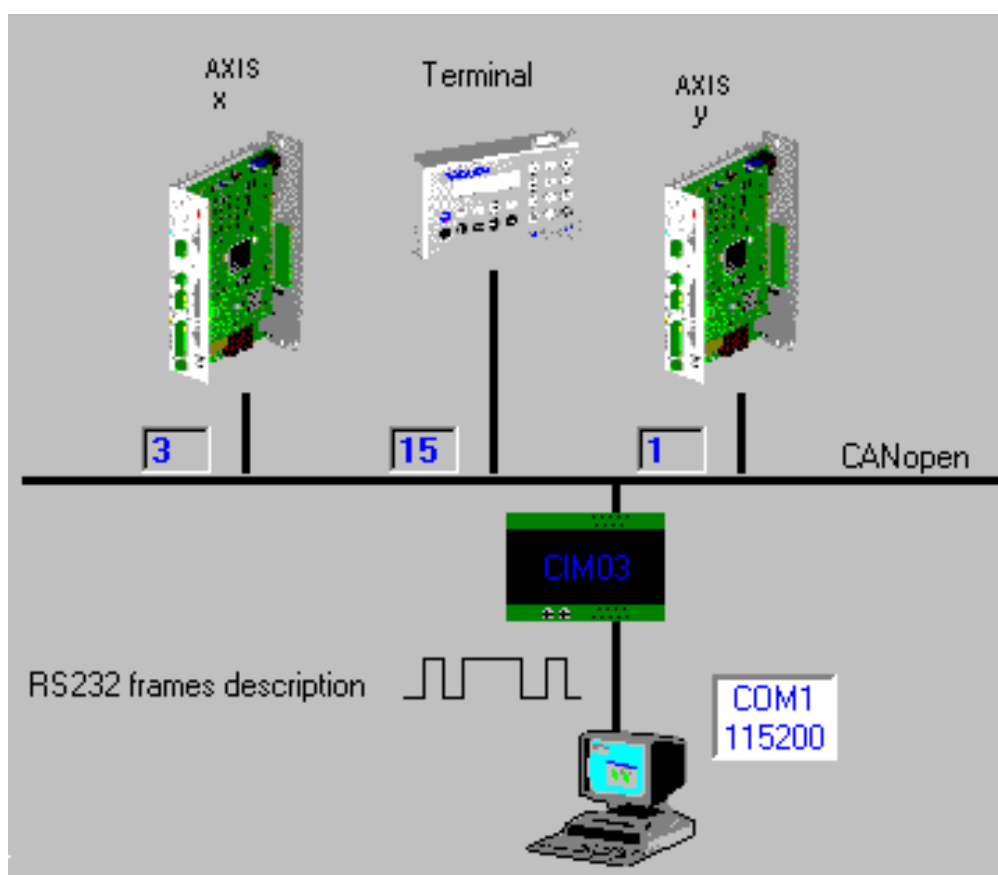
www.SSDdrives.com



DIGIVEX MOTION

CAN bus access via CIM03

PVD 3533 GB – 11/2003



PRODUCT RANGE

1 - « BRUSHLESS » SERVODRIVES

TORQUE OR POWER RANGES

- **BRUSHLESS SERVOMOTORS, LOW INERTIA, WITH RESOLVER**
 Very high torque/inertia ratio (high dynamic performance machinery):
 - ⇒ NX -HX - HXA 1 to 320 N.m
 - ⇒ NX - LX 0.45 to 64 N.m
 High rotor inertia for better inertia load matching:
 - ⇒ HS - LS 3.3 to 31 N.m
 Varied geometrical choice :
 - ⇒ short motors range HS - LS 3.3 to 31 N.m
 - ⇒ or small diameter motors : HD, LD 9 to 100 N.m
 Voltages to suit different mains supplies :
 - ⇒ 230V three-phase for «série L - NX»
 - ⇒ 400V, 460V three-phase for «série H - NX»
- **"DIGIVEX DRIVE" DIGITAL SERVOAMPLIFIERS**
 - ⇒ SINGLE-AXIS DSD
 - ⇒ COMPACT SINGLE-AXIS DμD, DLD
 - ⇒ POWER SINGLE-AXIS DPD
 - ⇒ MULTIPLE-AXIS DMD
- **"PARVEX MOTION EXPLORER" ADJUSTING SOFTWARE**

2 - SPINDLE DRIVES

- **SPINDLE SYNCHRONOUS MOTORS**
 - ⇒ "HV" COMPACT SERIES
 - ⇒ "HW" ELECTROSPINDLE, frameless, water-cooled motor From 5 to 110 kW
up to 60,000 rpm
- **"DIGIVEX" DIGITAL SERVOAMPLIFIERS**

3 - DC SERVODRIVES

- **"AXEM", "RS" SERIES SERVOMOTORS** 0.08 to 13 N.m
- **"RTS" SERVOAMPLIFIERS**
- **"RTE" SERVOAMPLIFIERS** for DC motors + resolver giving position measurement

4 - SPECIAL ADAPTATION SERVODRIVES

- **"EX" SERVOMOTORS** for explosive atmosphere
- **"AXL" COMPACT SERIES SERVOREDUCTORS** 5 to 700 N.m

5 - POSITIONING SYSTEMS

- **Numerical Controls « CYBER 4000 »** 1 to 4 axes
- **"CYBER 2000" NC** 1 to 2 axes
- **VARIABLE SPEED DRIVE - POSITIONER**
 - ⇒ SINGLE-AXIS DSM
 - ⇒ POWER SINGLE-AXIS DPM
 - ⇒ MULTIPLE-AXIS DMM
- **ADJUSTMENT AND PROGRAMMING SOFTWARE PARVEX MOTION EXPLORER**

CONTENTS

1. GENERAL POINTS	2
1.1 List of published DIGIVEX MOTION manuals	2
1.2 CIM03 in Stand_Alone mode	2
2. FRAME FORMAT	3
2.1 Symbols used	3
2.2 Serial link configuration	3
2.3 Frame format to be sent over serial link	3
2.3.1 General frame format	3
2.3.2 Frame check-sum computation method	4
2.4 Description of error codes returned	5
2.4.1 Errors specific to the CANopen network	5
2.4.2 Other possible error messages	5
2.5 Reading a drive parameter	6
2.6 Reading a drive parameter	6
2.7 Changing baud rate on RS232	7
2.8 Changing baud rate on CAN	7
3. SPECIMEN PROGRAMS	8
3.1 Reading the <i>moving</i> parameter	8
3.2 Writing the <i>ui0 = 300d</i> parameter	9
3.3 Writing the <i>posr = -12.5</i> parameter	11

Characteristics and dimensions subject to change without notice

YOUR LOCAL CORRESPONDENT

SSD Parvex SAS

8 Avenue du Lac / B.P 249 / F-21007 Dijon Cedex
Tél. : +33 (0)3 80 42 41 40 / Fax : +33 (0)3 80 42 41 23
www.SSDdrives.com

1. GENERAL POINTS

1.1 List of published DIGIVEX MOTION manuals

◆ DIGIVEX Single Motion (DSM) User Manual	(DSM)	PVD3515
◆ DIGIVEX Power Motion (DPM) User Manual	(DPM)	PVD3522
◆ DIGIVEX Multi Motion (DMM) User Manual	(DMM)	PVD3523
◆ DIGIVEX Motion - CANopen		PVD3518
◆ DIGIVEX Motion - Profibus		PVD3554
◆ PME-DIGIVEX Motion Adjustment Manual		PVD3516
◆ DIGIVEX Motion Directory of Variables		PVD3527
◆ DIGIVEX Motion Programming		PVD3517
◆ DIGIVEX Motion - Cam Function		PVD3538
◆ PME Tool kit User and Commissioning Manual		PVD3528
◆ CANopen - CAN Bus Access via CIM03		PVD3533
◆ CANopen - Remote control using PDO messages		PVD3543
◆ "Block Positioning" Application Software		PVD3519
◆ "Fly shear linear cutting" software application		PVD3531
◆ "Rotary blade cutting" software application		PVD3532

1.2 CIM03 in Stand_Alone mode

DIGIVEX MOTION positioner drive parameter setting and programming is done via the CAN bus (CANopen protocol). It is possible to communicate with these drives:

- by sending CANopen commands directly via a CAN card,
- by using CAN-RS232 interface cards (CRS232 or CIM03).

CRS232 or CIM03 interface cards receive messages via an RS232 serial link (please refer to DIGIVEX Motion – CANopen PVD 3518). They convert these messages into the corresponding CANopen format and so allow communication with the DIGIVEX MOTION devices. This is the principle behind PME-MOTION software which can be used for parameter setting or programming of DIGIVEX MOTION devices from a PC via a CAN-RS232 interface.

It is then possible in certain specific applications to use a CIM03 interface card in **stand_alone** mode (switch next to socket) to control a DIGIVEX MOTION drive no longer from a PC but from an automatic controller or any other control card with an RS232 serial link. This requires low-level programming of the RS232 frames.

The purpose of this document is to provide the information required for programming RS232 frames so as to go and read or write the drive parameters. Section 2 describes the general RS232 frame format. The variables are all described in the "DIGIVEX MOTION Directory of Variables" PVD 3527.

Section 3 gives three examples of programming:

- reading the **moving** parameter
- writing the **ui0** parameter
- writing the **posr** parameter

2. FRAME FORMAT

2.1 Symbols used

In the frame description provided in the remainder of the document, the various fields are coded by bytes. A byte is made up of 8 bits and is used for encoding an integer value from 0 to 255. The following notation is used:

- Decimal format: **130** or **130d**
- Hexadecimal format: **82h** or **0x82** or **\$82**

2.2 Serial link configuration

- 8 data bits
- 1 stop bit
- no parity bit
- authorised baud rates: 9600, 19200, 57600 et 115200 bauds

The CIM03 module starts up with a baud rate of 9600 bauds on the serial link. To increase this baud rate, see Section 2.7.

2.3 Frame format to be sent over serial link

The general frame format is described below. Any frame sent to the interface card is accepted by a frame matching this format.

2.3.1 General frame format

Message size	Subscriber	Application identifier	Operation code	Message control	Parameter number	Parameter value	Check-Sum
1 byte	1 byte	1 byte	1 byte	2 bytes	3 bytes	up to 244 bytes	2 bytes

- **Message size:** Operation to be performed on CAN or in the CRS232 or CIM03 module connected
- **Subscriber** CAN subscriber number (1...64) of drive
- **Application identifier** Unique number identifying the issuing application: 0 - 255
- **Operation code** Operation to be performed on CAN or in the CRS232 or CIM03 module connected

Transmission : **\$0B** (11d) : Read a parameter
 \$0C (12d) : Write a parameter
 \$30 (48d) : Change CAN baud rate
 \$40 (64d) : Change RS232 baud rate

Reception (acceptance):

- \$40** (64d) : Acceptance of drive parameter read.
The parameter value is sent to the "parameter value" field.
- \$60** (96d) : Acceptance of drive parameter write.
- \$80** (128d) : Non acceptance of read or write (problem on CAN)
- \$E0** (224d) : Non acceptance specific to interface card (RS232 problem)
- \$F0** (240d) : Acceptance of a function (specific to interface card)

- **Message control** 2 bytes for processing messages split into several frames

This function is used internally to transfer programs of more than 244 bytes requiring several frames. In you applications, encode: **\$00, \$01**.

Remark: This field disappears for operation codes other than \$0B (read) and \$0C (write).

- **Parameter number** 3 bytes giving the parameter number to be written/read

Most significant	Least significant
Index	Sub-index

You will find the index and sub-index for the parameter you wish to use in the DIGIVEX MOTION Directory of Variables PVD 3527.

Remark: This field disappears for operation codes other than \$0B (read) and \$0C (write).

- **Parameter value**
 - * empty for read request
 - * value read for read acceptance
 - * empty for write acceptance
 - * value to be written for write request
 - * error code for non acceptance
- **Check-Sum** 2 bytes for validating frame upon reception (sum of all frame bytes except those of check-sum)

2.3.2 Frame check-sum computation method

Example of writing integer value 32767 to user variable ui1 (CAN index: 11009, sub-index CAN:0):

Message size	Subscriber	Application identifier	Operation code	Message control	Parameter number		Parameter value				Check-Sum															
14	33	1	12	1	11009	0	32767 = 7FFF				xyxy															
\$0E	\$21	\$1	\$0C	\$00 \$01	\$2B \$01	\$00	\$FF	\$7F	\$00	\$00	\$aa	\$BB														
14	+	33	+	1	+	12	+	0	+	1	+	43	+	1	+	0	+	255	+	127	+	0	+	0	=	487

Check-sum = 14d + 33d + 1 + 12d + 0 + 1 + 43d + 1 + 0 + 255d + 127d + 0 + 0
 = 487d
 = \$01E7

Check-Sum most significant bytes: \$aa = \$01
 Check-Sum least significant bytes \$BB = \$E7

2.4 Description of error codes returned

2.4.1 Errors specific to the CANopen network

As described in the previous section, if an error occurs on the CAN bus, a failed to accept message is returned via the interface card. This message contains **0x80** as the identification byte and the error type is described in the "parameter value" field.

The error type is encoded on **4 bytes** under the CANopen standard in the following format:

AC0	AC1	EC0	CL1
Additional code, low byte	Additional code, high byte	Error Code	Error Class

Additional code (AC0)	Meaning
\$00	No specific reason for this error
\$10	Parameter not valid
\$11	Sub-index is non-existent data
\$12	Service parameter too long
\$13	Service parameter too short
\$20	Service cannot be performed
\$21	because of local control
\$22	because of current device status
\$30	Parameter value beyond limits
\$31	Parameter value too high
\$32	Parameter value too low
\$36	Maximum parameter value is smaller than minimum value
\$40	Incompatible with other values
\$41	data does not fit PDO format
\$42	PDO length exceeded
\$43	General incompatibility of parameter
\$47	General incompatibility internal to device

Error Class (CL1)	Error Code (EC0)	Possible meaning
\$05 Service Error	\$03 Error on parameter	Toggle bit not alternated
	\$04 Illegal parameter	Time-out value reached
\$06 Access Error	\$01 Access to object prohibited	Write Read-Only parameter or read Write-Only parameter
	\$02 Non-existent object	Object not found in dictionary
	\$06 Hardware fault	Access refused because of hardware fault
	\$07 Type conflict	Non-compatible data types
	\$09 Inconsistent object attribute	Non-existent sub-index
\$08 Other Error	\$00	Transfer interrupted by user

2.4.2 Other possible error messages

- * **08d 0d 0d 224d 0d 0d 2d 0d 234d**
Time-out error on RS232 (e.g. incomplete frame)
- * **08d 0d 0d 224d 0d 0d 1d 0d 233d**
Check-Sum error on RS232 (check-sum wrongly calculated)
- * **14d subscriber appli 128d 0d 1d index index sub-index 0d 0d 2d 8d cs1 cs2**
Time-out error on CAN (e.g. drive not connected)

2.5 Reading a drive parameter

The frame transmitted must be in the following format:

Message size	Subscriber	Application identifier	Operation Code	Message control	Parameter number	Check-Sum
\$14 (20d)	1 byte	1 byte	\$0B (11d)	\$00 \$01	3 bytes	2 bytes

If read successfully, the acceptance frame should be as follows:

Message size	Subscriber	Application identifier	Operation Code	Message control	Parameter number	Parameter value	Check-Sum
1 byte	1 byte	1 byte	\$40	\$00 \$01	3 bytes	N bytes	2 bytes

where : Subscriber : the subscriber number sent on transmission
 Application identifier: the identifier sent on transmission
 Parameter number: the parameter requested on transmission
 Parameter value: the value read

See the DIGIVEX MOTION Directory of Variables PVD 3527 to find out the format of the value read. The value returned is encoded as follows:

- * Boolean : 4 bytes : \$01 \$00 \$00 \$00 or \$00 \$00 \$00 \$00
- * integer: 4 bytes (least significant to most significant)
e.g., 65800d = 00010108h is encoded \$08 \$01 \$01 \$00
- * floating (coded according to the IEEE754 standard): 4 bytes (lightweight to heavyweight)
- * double (widespread precision floating coded according to the IEEE754 standard): 8 bytes (lightweight to heavyweight)
- * character string: 16 bytes (ASCII character representation)

IMPORTANT : For the parameter value, the least significant bytes are transmitted first. Transmission ends with the most significant byte.

2.6 Reading a drive parameter

The frame transmitted must be in the following format (See DIGIVEX MOTION Directory of Variables PVD 3527 to find out the format of the value to be written). The format of the value to be sent is described in Section 2.5.

Message size	Subscriber	Application identifier	Operation Code	Message control	Parameter number	Parameter value	Check-Sum
1 byte	1 byte	1 byte	\$0c (12d)	\$00 \$01	3 bytes	N bytes	2 bytes

If written successfully, the acceptance frame should be as follows:

Message size	Subscriber	Application identifier	Operation Code	Message control	Parameter number	Check-Sum
0x14 (20d)	1 byte	1 byte	\$60 (96d)	\$00 \$01	3 bytes	2 bytes

where : Subscriber : the subscriber number sent on transmission
 Application identifier: the identifier sent on transmission
 Parameter number: the parameter requested on transmission

2.7 Changing baud rate on RS232

The frame transmitted must be in the format as follows:

Message size	Subscriber	Application identifier	Operation Code	Parameter number	Check-Sum
\$08	\$00	\$nn	\$40	\$val \$00 \$00	2 bytes

The value of the \$val parameter gives the baud rate to be applied to the RS232 serial link.

\$val = \$00	→	Baud rate = 9600 bauds
\$val = \$01	→	Baud rate = 19200 bauds
\$val = \$02	→	Baud rate = 57600 bauds
\$val = \$03	→	Baud rate = 115200 bauds

The CIM03 accepts at the former baud rate:

Message size	Subscriber	Application identifier	Operation Code	Parameter number	Check-Sum
\$08	\$00	\$nn	\$F0	\$val \$00 \$00	2 bytes

Messages can then be sent at the new baud rate.

2.8 Changing baud rate on CAN

When the interface starts up, the baud rate programmed on CAN is 1 Mbauds.

This baud rate must be reduced for networks more than 25 m in total.

See the DIGIVEX MOTION OPTIONS - µVision -RS232 - CAN instructions PVD3518 to find out the right baud rate for your network.

The message to be transmitted to the interface card is as follows:

Message size	Subscriber	Application identifier	Operation Code	Parameter number	Check-Sum
\$08	\$00	\$nn	\$30	\$val \$00 \$00	2 bytes

The \$val parameter gives the baud rate to be applied on the CAN network:

\$val = \$00	→	Baud rate = 10 kbauds
\$val = \$01	→	Baud rate = 20 kbauds
\$val = \$02	→	Baud rate = 50 kbauds
\$val = \$03	→	Baud rate = 125 kbauds
\$val = \$04	→	Baud rate = 250 kbauds
\$val = \$05	→	Baud rate = 500 kbauds
\$val = \$06	→	Baud rate = 1 Mbaud

The interface card must accept with the following message.

Message size	Subscriber	Application identifier	Operation Code	Parameter number	Check-Sum
\$08	\$00	\$nn	\$F0	\$val \$00 \$00	2 bytes

3. SPECIMEN PROGRAMS

This section describes three examples describing how to read and write three parameters of a positioner drive whose subscriber number is 12.

3.1 Reading the *moving* parameter

(*moving* = motion ongoing information)

* Get index and sub-index:

From "Directory of Variables":

	Decimal	Hexadecimal	
* index	= 11549d	\$2D1D	
* sub-index	= 00d	\$00	
* <u>Frame composition:</u>			
	Decimal	Hexadecimal	
* Message size	= 10d	\$0A	
* Subscriber	= 12d	\$0C	
* Sender application identifier	= 01d	\$01	(for example)
* Operation code	= 11d	\$0B	(read)
* Message control	= 00d ; 01d	\$00 ; \$01	
* index	= 45d ; 29d	\$2D ; \$1D	
* Sub-index	= 00d	\$00	
* Most significant check sum	= 00d	\$00	
* Least significant check sum	= 109d	\$6D	

$$(Csum = 10 + 12 + 1 + 11 + 0 + 1 + 45 + 29 + 0 = 109d = \$6D)$$

* Expected result:

	Decimal	Hexadecimal	
* Message size	= 14d	\$0E	
* Subscriber	= 12d	\$0C	
* Sender application identifier	= 01d	\$01	
* Operation code	= 64d	\$40	(read successful)
* Message control	= 00d ; 01d	\$00 ; \$01	
* Index	= 45d ; 29d	\$2D ; \$1D	
* Sub-index	= 00d	\$00	
* Data	= 01d 00d 00d 00d	\$01000000	if moving
	= 00d 00d 00d 00d	\$00000000	if not moving
* Most significant check sum	= 00d	\$00	
* Least significant check sum	= 167d	\$A7	if moving
	= 166d	\$A6	if not moving

* Delphi specimen program:

```
function interrogation_mouvement : boolean;
var
  recept : string;
begin
  // send "read moving" frame over serial link
  LiaisonSerie1.TransmitCar(chr(10));
  LiaisonSerie1.TransmitCar(chr(12));
  LiaisonSerie1.TransmitCar(chr(1));
  LiaisonSerie1.TransmitCar(chr(11));
  LiaisonSerie1.TransmitCar(chr(0));
  LiaisonSerie1.TransmitCar(chr(1));
  LiaisonSerie1.TransmitCar(chr(45));
  LiaisonSerie1.TransmitCar(chr(29));
  LiaisonSerie1.TransmitCar(chr(0));
  LiaisonSerie1.TransmitCar(chr(0));
  LiaisonSerie1.TransmitCar(chr(109));

  // wait for acceptance frame
  Attente_trame;

  // recover acceptance frame
  recept := LiaisonSerie1.LireBufferRX(15);
  if (ord(recept[4]) = 64) then // read successful
    begin
      if (ord(recept[10]) = 0) then
        begin
          // not moving
          result := false;
        end
      else
        begin
          // not moving
          result := true;
        end;
      end;
    end;
  end;
```

3.2 Writing the ui0 = 300d parameter

* Get index and sub-index:

From "Directory of Variables":

	Decimal	Hexadecimal
* index	= 11008d	\$2B00
* sub-index	= 00d	\$00

Value **300d** will be encoded in integer format over 4 bytes: (**300d = \$012C**)
\$2C \$01 \$00 \$00

* Frame composition:

	Decimal	Hexadecimal	
* Message size	= 14d	\$0E	
* Subscriber	= 12d	\$0C	
* Sender application identifier	= 01d	\$01	(for example)
* Operation code	= 12d	\$0C	(write)
* Message control	= 00d; 01d	\$00 ; \$01	
* Index	= 43d ; 00d	\$2B ; \$00	
* Sub-index	= 00d	\$00	
* Data (300)	= 44d 01d 00d 00d	\$2C010000	
* Most significant check sum	= 00d	\$00	
* Least significant check sum	= 128d	\$80	

(Csum = 14 + 12 + 1 + 12 + 0 + 1 + 43 + 0 + 0 + 44 + 1 = 128d = \$80)

* Expected result:

	Decimal	Hexadecimal	
* Message size	= 10d	\$0A	
* Subscriber	= 12d	\$0C	
* Sender application identifier	= 01d	\$01	
* Operation code	= 96d	\$60	(write successful)
* Message control	= 00d; 01d	\$00 ; \$01	
* Index	= 43d ; 00d	\$2B ; \$00	
* Sub-index	= 00d	\$00	
* Most significant check sum	= 00d	\$00	
* Least significant check sum	= 163d	\$A3	

* Delphi specimen program:

```
// send "write ui0=300" frame over serial link
LiaisonSerie1.TransmitCar(chr(14));
LiaisonSerie1.TransmitCar(chr(12));
LiaisonSerie1.TransmitCar(chr(1));
LiaisonSerie1.TransmitCar(chr(12));
LiaisonSerie1.TransmitCar(chr(0));
LiaisonSerie1.TransmitCar(chr(1));
LiaisonSerie1.TransmitCar(chr(43));
LiaisonSerie1.TransmitCar(chr(0));
LiaisonSerie1.TransmitCar(chr(44));
LiaisonSerie1.TransmitCar(chr(1));
LiaisonSerie1.TransmitCar(chr(0));
LiaisonSerie1.TransmitCar(chr(0));
LiaisonSerie1.TransmitCar(chr(0));
LiaisonSerie1.TransmitCar(chr(128));

// wait for acceptance frame
Attente_trame;

// recover acceptance frame
recept := LiaisonSerie1.LireBufferRX(11);
if (ord(recept[4]) = 96) then // write successful
begin
  // action to be done
end;
```

3.3 Writing the posr = -12.5 parameter

* Get index and sub-index:

From "Directory of Variables"

	Decimal	Hexadecimal
* index	= 11590d	\$2D46
* sub-index	= 00d	\$00

Value **-12.5** will be encoded in double format over 8 bytes:

\$C0 (192d) \$29 (41d) \$00 \$00 \$00 \$00 \$00 \$00

* Frame composition:

	Decimal	Hexadecimal	
* Message size	= 18d	\$12	
* Subscriber	= 12d	\$0C	
* Sender application identifier	= 01d	\$01	(for example)
* Operation code	= 12d	\$0C	(write)
* Message control	= 00d ; 01d	\$00 ; \$01	
* Index	= 45d ; 70d	\$2D ; \$46	
* Sub-index	= 00d	\$00	
* Data (-12.5)	= 00d 00d 00d 00d	\$00 00 00 00	
	= 00d 00d 41d 192d	00 00 29 C0	
* Most significant check sum	= 01d	\$01	
* Least significant check sum	= 136d	\$88	

(Csum = 18 + 12 + 1 + 12 + 0 + 1 + 45 + 70 + 0 + 41 + 192 = 392d = \$0188)

* Expected result:

	Decimal	Hexadecimal	
* Message size	= 10d	\$0A	
* Subscriber	= 12d	\$0C	
* Sender application identifier	= 01d	\$01	
* Operation code	= 96d	\$60	(write successful)
* Message control	= 00d ; 01d	\$00 ; \$01	
* Index	= 45d ; 70d	\$2D ; \$46	
* Sub-index	= 00d	\$00	
* Most significant check sum	= 00d	\$00	
* Least significant check sum	= 235d	\$EB	

Important remark: this response means the drive has accepted the command but that it has not necessarily been completed, particularly in this example where movement is made.

To check the motion is completed, read the **moving** parameter (see earlier example).

Notice too that data are transmitted beginning with the least significant byte and ending with the most significant.

* Delphi specimen program:

```
// send "write posr = -12.5" frame over serial link
LiaisonSerie1.TransmitCar(chr(18));
LiaisonSerie1.TransmitCar(chr(12));
LiaisonSerie1.TransmitCar(chr(1));
LiaisonSerie1.TransmitCar(chr(12));
LiaisonSerie1.TransmitCar(chr(0));
LiaisonSerie1.TransmitCar(chr(1));
LiaisonSerie1.TransmitCar(chr(45));
LiaisonSerie1.TransmitCar(chr(70));
LiaisonSerie1.TransmitCar(chr(0));
LiaisonSerie1.TransmitCar(chr(0));
LiaisonSerie1.TransmitCar(chr(0));
LiaisonSerie1.TransmitCar(chr(0));
LiaisonSerie1.TransmitCar(chr(0));
LiaisonSerie1.TransmitCar(chr(0));
LiaisonSerie1.TransmitCar(chr(0));
LiaisonSerie1.TransmitCar(chr(41));
LiaisonSerie1.TransmitCar(chr(192));
LiaisonSerie1.TransmitCar(chr(1));
LiaisonSerie1.TransmitCar(chr(136));

// wait for acceptance frame
Attente_frame;

// recover acceptance frame
recept := LiaisonSerie1.LireBufferRX(11);
if (ord(recept[4]) = 96) then           // write successful
    begin
        // wait for end of motion
        while not(interrogation_mouvement);
        // motion completed
    end;
```